

Authority ring periodically load collection for load balancing of cluster system

Sharada Santosh Patil^{1,2,*}, Arpita N. Gopal²

¹MCA, Deptt. SIBAR Kondhwa, Pune, Maharashtra, INDIA

²Director MCA, SIBAR, Kondhwa, Pune, Maharashtra INDIA

Email address:

sharada_jadhao@yahoo.com(S. S. Patil), arpita.gopal@gmail.com(A. N. Gopal)

To cite this article:

Sharada Santosh Patil, Arpita N. Gopal. Authority Ring Periodically Load Collection for Load Balancing of Cluster System. *American Journal of Networks and Communications*. Vol. 2, No. 5, 2013, pp. 133-139. doi: 10.11648/j.ajnc.20130205.13

Abstract: Now a day clusters are more popular because of their parallel processing and super computing capabilities. Generally complicated processes needs more amount of time to run on single processor but same can give quick result on clusters because of their parallel execution capabilities due to their compute nodes. Typically this performance depends on which load balancing algorithm is running on the clustered system. The parallel programming on the cluster can achieve through message passing interface or application programming interface (API). Though the load balancing algorithm distributing load among the compute nodes of the clusters, it needs parallel programming, hence MPI library plays very important role to build new load balancing algorithm. The workload on a cluster system can be highly variable, increasing the difficulty of balancing the load across its compute nodes. This paper proposes new dynamic load balancing algorithm, which is implemented on Rock cluster and maximum time it gives the better performance as compares with previous dynamic load balancing algorithm.

Keywords: MPI, Parallel Programming, HPC Clusters, DLBA ARPLCLB, ARPLC

1. Introduction

Scheduling of processes among the compute nodes of the cluster system has always been an important and challenging area of research. The research is more challenging because of the various factors involved in implementing a load balancing algorithm in clustered system. Some of these influencing factors are the parallel workload, presence of any sequential and/or interactive jobs, native operating system, node hardware, network interface, network, and communication software. The main objective of load balancing algorithm is to speed up the system and enhance super computing power within the clustered system. There are two main types of performing load balancing – static policy and dynamic policy. [9].

1.1. Static Load Balancing Policies

Static load balancing policies judge system and application status statically and apply this information in decision making. The two renowned static policies are mentioned below.

- (1). Load-dependent static policy
- (2). Speed-weighted random splitting policy[8].

1.2. Dynamic Load Balancing Policies

In dynamic policies, workload is distributed among the processors at run time. New processes are assigned to the processors based on the runtime information that is collected from each node. If each node in the system becomes overloaded, the task that causes this overloading should be transferred to an under loaded node and run there. Although the dynamic policies have many benefits and can adopt with the current state of the system, sometimes they will incur extra overhead on the system because of the process migration and reservation resources of the system for collecting the information of the current status of the system. The information exchange policy can obey one of the following policies [8].

- (1). Periodic policies
- (2). Demand-driven policies
- (3). State-change-driven policies

Load imbalance has three main sources; application imbalance, workload imbalance and heterogeneity of hardware resources. Application imbalance occurs when different parallel threads of computation take varying times to complete the super step. [20] [7]

Load balancing in the application level concentrates on minimizing the completion time of an application while load balancing in the system level is known as a distributed scheduling that is used for maximizing the throughput or utilization rate of the nodes. [1]

1.3. Centralized and Decentralized Scheme

In Centralized schemes, a central manager collects the information of each node and performs the decision making based on overall knowledge of the system. In a distributed load balancing scheme, each node makes decision based on its local knowledge. A neighbour based scheme is a distributed scheme that makes efficient load decisions without having any overall knowledge about the system.

In Decentralized scheme each node can participate in load balancing decision, but distribution of load is depend upon complete knowledge i.e. load of each node. Here, each node uses same load distribution policy. Hence accordingly it distributes load using process transfer policy. (Parimah Mohammadpour, Mohsen Sharifi, Ali Paikan-2008) (Janhavi B,Sunil Surve ,Sapna Prabhu-2010) [8][5]

1.4. Pre-Emptive and Non Pre-Emptive Load Balancing

In non pre-emptive load balancing approach, only those processes of heavily loaded node are migrated to lightly loaded node whose execution has not started yet. In other words only new born processes can be migrated, running processes cannot be migrated in this system. In pre-emptive load balancing approach processes in any state of heavily loaded nodes are migrated to lightly loaded node where the process state could be new born or running or waiting. [17][19][20].

2. The Message Passing Interface (MPI)

Message Passing Interface (MPI) is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers. The standard defines the syntax and semantics of a core of library routines useful to a maximum size of users writing portable message-passing programs in Fortran 77 or the C programming language. According to R. Butler and E. Lusk P4 [2] is a third-generation parallel programming library, including both message passing and shared-memory components, portable to a great many parallel computing environments, including heterogeneous networks. Although p4 contributed much of the code for TCP/IP networks and shared-memory multiprocessors for the early versions of MPICH, most of that has been rewritten. P4 remains one of the “devices” on which MPICH can be built (see Section 4.1, but in most cases more customized alternatives are available.

Chameleon Written by W.D. Gropp and B. Smith [3][4][18] is a high-performance portability package for message passing on parallel supercomputers. It is implemented as a thin layer (mostly C macros) over vendor

message-passing systems (Intel’s NX, TMC’s CMMD, IBM’s MPL) for performance and over publicly available systems (~4 and PVM) for portability. A substantial amount of Chameleon technology is incorporated into MPICH.

3. The Idea of New Research

As it has been said before, the distribution rule can be based on remote execution or in process migration. Many authors have discussed about the benefits and drawbacks of process migration for the algorithm distribution rule.

Some authors have concluded that migration cannot provide better performance than other alternatives like remote execution due to its costs, while others have argued that even when the costs of migration is high, it can significantly improve systems’ performance, especially when the systems are highly variable.[5]

The dynamic load balancing algorithm mainly removes the bottle necks presented by the static co-scheduling approach thus making the cluster co-scheduling scalable. But it presents a larger communication overhead as compared to static load balancing algorithm because dynamic load balancing algorithm performs process migration. In a centralized load balancing algorithm, load is distributed uniformly among the processors. The only disadvantage is maximum time of the central processor is wasted in load balancing rather than process execution. Hence performance of the server decreases. A decentralized load balancing algorithm decision of load distribution is taken by all the node hence, each node has load of other nodes and communication overhead increases tremendously.

In order to balance the load uniformly over a cluster system, one has to choose a mix of centralized and decentralized approach.

This communication overhead and load balancing time depends upon the approach selected in the algorithm. Those approaches are given below;

The load balancing algorithm for the clusters can be made more robust by scheduling all jobs irrespective of any constraints so as to balance the load perfectly. This work is extended to develop a new algorithm that modify dynamic decentralized approach so as to reduce the communication overhead as well as reduce migration time and also make it scalable.

4. Authority Ring Periodically Load Collection for Load Balancing Algorithm (ARPLCLB)

As said in above discussion, this algorithms mix two approaches - centralized and decentralized. The authority packet is circulated among the compute nodes. Whenever system is completely imbalanced, any lowly loaded processor can pick up this authority packet and get authority to become master node. Master node is responsible to balance the system. Every compute node can broadcast load

to others at certain period such that they can evaluate their current state to find whether the system is balanced or imbalanced. Hence the name of this algorithm is Authority Ring Periodically Load Collection for Load Balancing Algorithm in short it is (ARPLCLB)

This section explains overall procedure, different policies used in the algorithm, Data structure used to build algorithm, and parallel algorithm

4.1. Overall Procedure

The Overall Procedure of this algorithm is given below;

Step 1: After completion of every ring period, every processor passes or broadcasts information packet to all processor which consists of;

1. Current status of the node
2. Current load of the node with load factor

Step 2: Every processor can store the current information as well as past information of all the processors.

Step 3: Every processor collects authority packet from previous processor and circulate it to next processor.

Step 4: When any Idle node or Low load node get authority packet then immediately it take the charge of master node and performs following activities;

1. Create workload Distribution table using following process criteria;
 - a. It chooses newly arrived processes. (That means new born Processes)
 - b. It chooses processes which needs 80 % time for execution.
2. Create Order packets according to Workload Distribution table
3. Send order packet of all node to that appropriate node.
4. After load distribution send authority packet to next node.

Step 5: As soon as any node gets order packet they should follow the order of order packet to perform process migration.

Step 6: Master node again starts authority ring means authority packet is circulated to each node of the LAN one by one again.

Step 7: Repeats Steps 1 to 7 till cluster is not shut down

5. Policies Used in ARPLCLB Algorithm

Following Policies used in proposed dynamic load balancing algorithm.

5.1. Load Information Policy

Load information serves as one of the most fundamental elements in the load balancing process. Every dynamic load balancing algorithm is based on some type of load information. According to this algorithm, each load of cluster system has some current state. These CPU states can be idle, lowly loaded, normal or heavily loaded CPU.

1. The CPU state can be idle state, if ready queue is empty

and it is not executing any process and hence 100 % memory is available.

$$\sum_{i=0}^{Q_{total}} P_i \approx 0 \text{ \& } MEM_{free} \approx 100\%$$

2. The CPU state can be low state, if total number of processes < (less than) LOW_LOAD_THRESHOLD_VALUE (L) * size of queue (Q_s) and more than 75 % memory (MEM_{free}) is available.

$$\sum_{i=0}^{Q_{total}} P_i \leq L * Q_s \text{ \& } 75\% \leq MEM_{free}$$

3. The CPU state can be normal state if total number of processes < (less than) NORMAL_LOAD_THRESHOLD_VALUE (N) * size of queue and 25 % to 75% memory is available.

$$\sum_{i=0}^{Q_{total}} P_i \leq N * Q_s \text{ \& } 25\% \leq MEM_{free} < 75$$

4. The CPU state can be heavy state if total number of processes > (greater than) NORMAL_LOAD_THRESHOLD_VALUE * Size of Queue and less than 25% memory is available.

$$\sum_{i=0}^{Q_{total}} P_i \leq N * Q_s \text{ \& } MEM_{free} < 25$$

5. The system balance depends on following criteria;
 - When all nodes are heavily loaded then system can be called as heavily balanced system.
 - When heavily loaded nodes are 1% to 85% and remaining are lowly loaded or idle or normal processors then system is imbalanced and need process migration
 - When no node is heavily loaded and may be idle or lowly loaded or normal loaded then system is called slightly balanced or slightly imbalanced system, and it does not require any process migration.

5.2. Information Exchange Policies of ARPLCLB

This information exchange policy depends on how node can be exchange load information with others. This algorithm uses periodic policy means it exchanges this load information after every certain number of time slot. This information policy is executed by all nodes.

5.3. Process Transfer Policies of ARPLCLB

Process can be transfer from heavily loaded processor to idle or lowly loaded or normally loaded processor. This policy is executed on master node. This policy can be mentioned as follows;

1. System is heavily balanced if all CPU in the clusters are heavily loaded then it execute delay of 1000 such that all CPU can execute their load to get relief from authority token ring.

2. When system is slightly balanced or slightly imbalanced, then system is in normal condition
3. When system is completely imbalanced then this algorithm decides decision of process migration.
4. For process transfer activity , it calculates the ideal load of each processor as follows;

$$Ideal_load = \frac{Total_System_load}{Total_Computenode_cluster}$$

5. It transfers total ideal load processes from heavily loaded processor to lightly loaded processor

5.4. Selection Policies of ARPLCLB

A selection policy decides which process is selected for transfer that means process migration. The chosen process could be a new process which has not started, that means, new born process or an old process which is already starting its execution. If it is old process then it should satisfy following condition.

1. If $(rbt/bt * 100 \geq 80)$ Process is selected for migration.

$$\frac{Remaining_Burst_Time}{Burst_Time} * 100 \geq 90? : Process_Migration$$

This selection policy is executed by master node.

5.5. Location Policies of ARPLC

Location policy decides selected processes for migration is migrated to which CPU For this activity , it select ideal CPU to migrate processes from heavily loaded CPU to lightly loaded CPU using following steps

1. Select heavy loaded CPU
2. Select first idle CPU
 - a. If found go to 3 else b
 - b. Select first low loaded CPU
 - i. If found go to step 3 else ii
 - ii. Select first normal loaded CPU
3. Select process for migration to selected CPU
4. Update load of that CPU
5. Repeat 3 and 4 till Load of CPU < ideal load of the system

This location policy is executed by master node.

6. Parallel Sub Algorithms of ARPLCLB)

This algorithm ARPLCLB is divided in to three parallel sub algorithms, that are;

1. Authority token ring with Periodically Load collection algorithm(ARPLC)
2. Load Distribution With Order Packet Algorithm(LDOP)
3. Process Migration Algorithm(PM)

6.1. Authority Token Ring with Periodically Load Collection Algorithm (ARPLC)

This algorithm is similar to musical ball passing game. Similar to musical ball authority packet is moved around the logical ring of the compute nodes of the cluster. This algorithm circulates authority tokens between the processors such that they can choose their new master node when cluster system is imbalanced. This algorithm collects load information periodically. The master node CPU can be responsible to start authority ring, in this it passes authority packet to next nearby neighbor. When system is imbalanced then any lowly loaded or idle node picks up this authority packet to become new master node. This node conveys this information to all using new master indication packet. This is explained in following figure 1

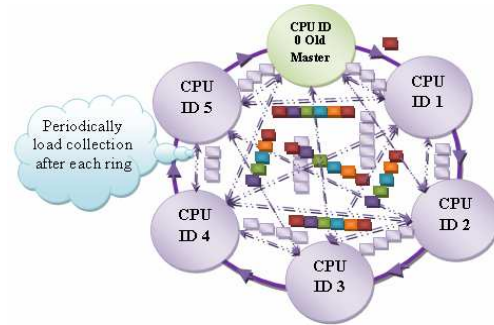


Figure 1: Authority token Ring with Periodically Load collection Algorithm (ARPLC)

6.2. Load Distribution with Order Packet Algorithm (LDOP)

When cluster system is imbalanced and new master node is decided by ARPLC parallel sub algorithm, then this new master node has load information of each node. Hence it decides load distribution using load information and distribution policy, process selection and location policy, accordingly creates order packets, and distributes or broadcasts this order packet to each node, and calls process migration algorithm which follows order of master node blindly. Maximum part of this LDOP algorithm is executed on master node and very minimum part of algorithm is executed on other compute node. This is explained in following figure 2.

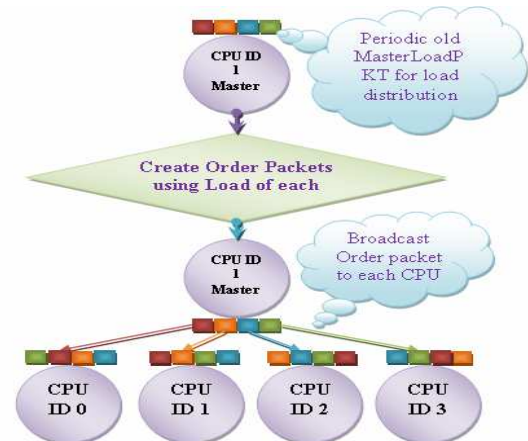


Figure 2: Load Distribution with Order Packet Algorithm (LDOP)

6.3. Process Migration of Algorithm (PM)

Once order packet is distributed by the new master node among all other compute nodes of the cluster then other compute nodes are automatically divided in to heavy load CPU group and idle or low load CPU group. And heavy loaded CPU transfers their own load to lightly loaded CPU. This is explained in figure 3. The load is transferred using two types of the packets , that means process control block packet whose size is fixed and then it transfers actual process to destination CPU such that it can be easily start its remaining execution on new processor.

When process migration is over it again execute the ARPLC parallel sub algorithm to monitor system imbalance and distribute authority packet among the compute nodes

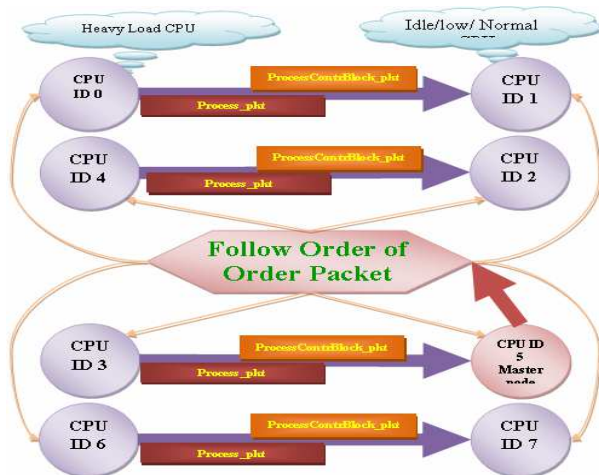


Figure 3: Process Migration of (PM) Algorithm

As it has been said before, the distribution rule can be based on remote execution or in process migration

7. Performance of the Authority Ring with Periodically Load Collection Algorithm

Table 6.3: Performance of Algorithm1 (ARPLCLB)

Iteration of Outer Loop	Total Process Migration	Total Number of Rings	New Master	Old master
1	9	2	2	0
2	9	1	2	2
3	8	6	2	2
4	8	2	0	2
5	8	1	0	0
6	8	4	3	0
7	9	4	0	3
8	8	4	3	0
9	8	1	0	3
10	8	4	3	0

For evaluating the performance of the above algorithm we implement algorithm run it on Rock cluster on centos operating system then it produces some output in data file. These data files are too lengthy hence only some result are given, following figure shows screen shot of the same during the execution.

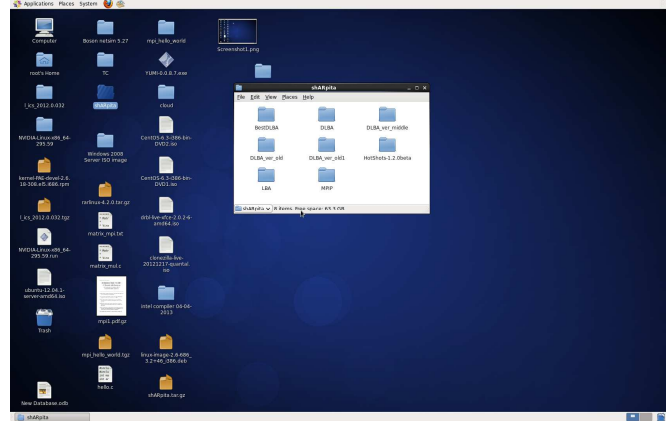


Figure 4: Screen shots of ARPLCLB(a)

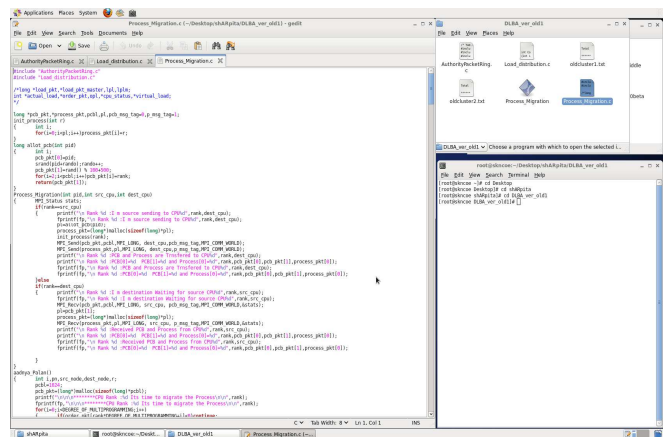


Figure 4: Screen shots of ARPLCLB(b)

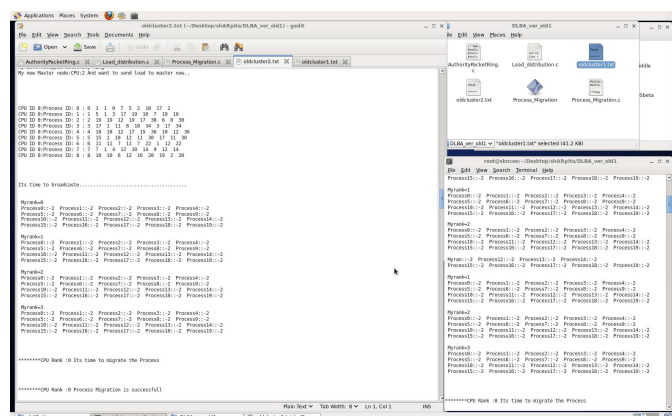


Figure 4: Screen shots of ARPLCLB(c)

The authority rings continues, means system is currently in balanced state. When system is in imbalanced stage, then it performs process migration. The graph of outer loop iteration against process migration is given in figure 5;

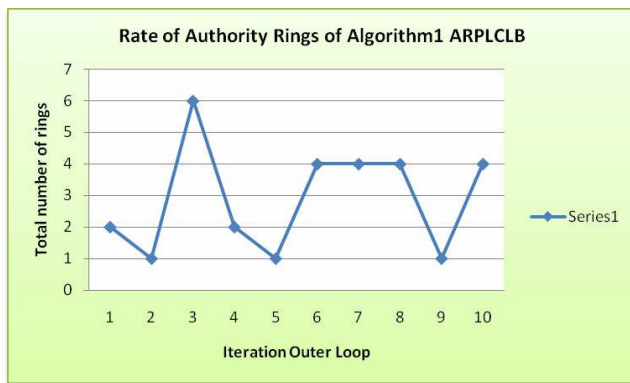


Figure 5: Performance of ARPLCLB using Process migration

As result shows, this algorithm migrates maximum number of processes in each number of iterations of the loop. Hence Maximum time of the CPU is wasting to perform process migration rather than process execution. Hence it is degradation of the system.

The graph shows total authority rings verses iteration of outer loop. When total authority rings are more means system is currently in balanced state.

The graph shows total authority rings verses iteration of outer loop. When total authority rings are more means system is currently in balanced state.

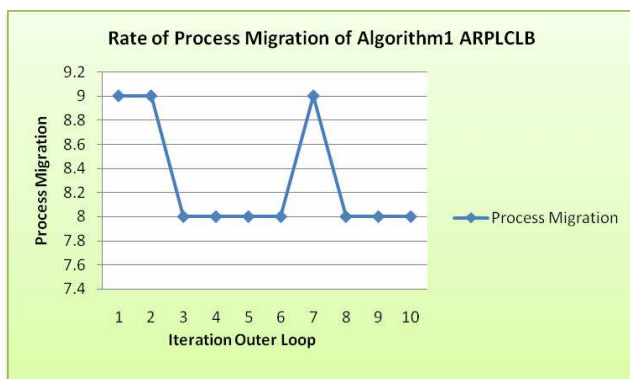


Figure 5: Performance of Algorithm 1 Using Total Rings (ARPLCLB)

The result of above graph states that every iteration of the ring migrates to too much processes. And this algorithm uses periodically load collection policy, hence, it uses too much communication overhead. The advantages and disadvantages of the algorithm are given below;

8. Conclusion

In order to balance the load uniformly over a cluster system, our proposed algorithm has used a mix of centralized, decentralized, approach. The performance of this algorithm gives better result many a time but due to heavy communication overhead and heavy process migration, affect the performance. All previous algorithms may either use centralized approach or decentralized approach. But this proposed algorithm uses both approaches. Hence in future work is extended to improve the result of

this algorithm. The load balancing algorithm for the clusters can be made more robust by scheduling all jobs irrespective of any constraints so as to balance the load perfectly

9. Future Enhancement

This work is extended to remove all disadvantages of this proposed algorithm. As well as policies used in this algorithm is also improved. In future, this work can be extended to develop a two new dynamic load balancing algorithm to modify dynamic decentralized approach so as to reduce the communication overhead as well as to reduce migration time and also make it scalable

Acknowledgment

We sincerely express our thanks to laboratory research cell of SIBAR-MCA and SKNCOE for their full support. We are very grateful to all friends who have directly and indirectly supported to this research work

References

- [1] Bernd F reisleben Dieter Hartmann Thilo Kielmann "Parallel Raytracing A Case Study on Partitioning and Scheduling on Workstation Clusters" 1997 Thirtieth Annual Hawaii International Conference on System Sciences.
- [2] Blaise Barney, (1994) Livermore Computing, MPI Web pages at Argonne National Laboratory <http://www-unix.mcs.anl.gov/mpi> "Using MPI", Gropp, Lusk and Skjellum. MIT Press
- [3] Erik D. Demaine, Ian Foster, Carl Kesselman, and Marc Snir. "Generalized Communicators in the Message Passing Interface" 2001 IEEE transactions on parallel and distributed systems pages from 610 to 616.
- [4] Hau Yee Sit Kei Shiu Ho Hong Va Leong Robert W. P. Luk Lai Kuen Ho "An Adaptive Clustering Approach to Dynamic Load balancing" 2004 IEEE 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)
- [5] Janhavi B, Sunil Surve, Sapna Prabhu "Comparison of load balancing algorithms in a Grid" 2010 International Conference on Data Storage and Data Engineering Pages from 20 to 23.
- [6] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker and J. Dongarra, (1996) MPI: The Complete Reference (MIT Press, Cambridge, MA, 1995). 828 W. Gropp et al./Parallel Computing 22 (1996) 789-828.
- [7] Marta Beltr'an and Antonio Guzm'an "Designing load balancing algorithms capable of dealing with workload variability" 2008 International Symposium on Parallel and Distributed Computing Pages from 107 to 114.
- [8] Parimah Mohammadpour, Mohsen Sharifi, Ali Paikan, "A Self-Training Algorithm for Load Balancing in Cluster Computing", 2008 IEEE Fourth International Conference on Networked Computing and Advanced Information Management, Pages from 104 to 110.

- [9] Paul Werstein, Hailing Situ and Zhiyi Huang „Load Balancing in a Cluster Computer“ 2006 Proceedings of the Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies.
- [10] Sharada Patil, Dr Arpita Gopal,[2012], Ms Pratibha Mandave “Parallel programming through Message Passing Interface to improving performance of clusters ” – International Doctoral Conference (ISSN 0974-0597) SIOM, Wadgoan Budruk in Feb 2013.
- [11] Sharada Patil, Arpita Gopal “Comparison of Cluster Scheduling Mechanism using Workload and System Parameters” 2011 ISSN 0974-0767 International journal of Computer Science and Application.
- [12] Sharada Patil, Arpita Gopal “STUDY OF DYNAMIC LOAD BALANCING ALGORITHMS FOR LINUX CLUSTERED SYSTEM USING SIMULATOR” 2011 ISSN 0974-3588 International journal of Computer Applications in Engineering Technology and Sciences.
- [13] Sharada Patil, Dr Arpita Gopal, [2011] “Study of Load Balancing Algorithms” – National Conference on biztech 2011, Selected as a best paper in the conference, got first rank to the paper, DICER, Narhe, Pune in year March 2011.
- [14] Sharada Patil, Dr Arpita Gopal, [2013] “Cluster Performance Evaluation using Load Balancing Algorithm” – INTERNATIONAL CONFERENCE ON INFORMATION COMMUNICATION AND EMBEDDED SYSTEMS ICICES 2013, 978-1-4673-5788-3/13/\$31.00©2013 IEEE (ISBN 978-1-4673-5786-9) Chennai, India in year Feb 2013.
- [15] Sharada Patil, Dr Arpita Gopal, [2012] “Need Of New Load Balancing Algorithms For Linux Clustered System” – International Conference on Computational techniques And Artificial intelligence (ICCTAI’2012) (ISBN 978-81-922428-5-9) Penang Malaysia in year Jan 2012.
- [16] Sharada Patil, Dr Arpita Gopal, [2013] “Enhancing Performance of Business By Using Extracted Supercomputing Power From Linux Cluster’s ” – International Conference on FDI 2013 (ISSN 0974-0597) SIOM, Wadgoan Budruk in Jan 2013
- [17] Sun Nian1, Liang Guangmin2 “Dynamic Load Balancing Algorithm for MPI Parallel Computing” 2010 Pages 95 to 99
- [18] William Gropp, Rusty Lusk, Rob Ross, and Rajiv Thakur (2005) “MPI Tutorials “ Retrieved from www.mcs.anl.gov/research/projects/mpi/tutorial Livermore Computing specific information:
- [19] Yanyong Zhang, Anand Sivasubramaniam, JoseÂ Moreira, and Hubertus Franke” Impact of Workload and System Parameters on Next Generation Cluster Scheduling Mechanisms” 2001 IEEE transactions on parallel and distributed systems Pages from 967 to 985.
- [20] Yongzhi Zhu Jing Guo Yanling Wang “Study on Dynamic Load Balancing Algorithm Based on MPICH” 2009 MPI_COMM_RANK: World Congress on Software Engineering. Pages from 103 to 107.